Znews

Vol. 1 No. 2, April 1989

The Fickle Finger of Fate

As I mentioned last month, the first issue of ***Znews*** was delayed by some pretty bizarre twists of fate. It was delayed even more after the originals left my little igloo - the new printshop/mailing house I contracted with informed me that their normal turnaround time is 20 working days. I'm afraid I didn't realize that ahead of time. I am therefore scrambling to get this issue out the door ASAP so you get it before next summer.

Ah, the tribulations of the publishing industry. I thank you for your patience.

And before I forget this: don't check your mailing labels for expiration dates, etc. yet. Since I'm pushing ahead production, I have had to postpone working on the "merge/purge" of Zedcor's database.

I got a nice note from Greg Branche, the late, great former Zedcor employee who helped develop the Apple II version of ZBasic (and worked on some of the others, too, I believe). Actually, Greg didn't die, but he did go to a heaven of sorts; he now works for Apple Computers, Inc. Anyway, Greg pointed out a few boo-boos I made last month.

The following applies to the Apple II ProDOS version. The rest of you should speed-read the next few paragraphs. Note the paragraph on how the compiler does arithmetic, though. That applies to everybody.

Take your pencils, friends, and turn to p.12 from the March issue. Look down to the GET_EOF# function, which looks like this:

```
LONG FN GET_EOF# (REF_NUM)
   POKE &1F00,2                    :REM
two parms for this call
   POKE &1F01, REF_NUM
   MACHLG &A9, &D1, &20, &0865 :   REM
make the call
   FILE_LEN# = PEEK WORD (&1F02) + PEEK
(&1F04) * 256 : REM length of file
 END FN = FILE_LEN#
```

If you compare GET_EOF# with the SET_MARK function I defined shortly thereafter, you'll see that I treated the three file length bytes beginning with &1F02 differently. Oops.

The fifth line of the function should read:

```
  FILE_LEN# = PEEK WORD (&1F02) + PEEK
(&1F04) * 65536.0 : REM length of file
```

At first I figured the 65536.0 had to be a typo in Greg's message. Why the decimal point? Well, here comes one of those many things I know that I once knew and promptly forgot. As Greg reminded me, those of us who are expatriates from other BASICs need to remember how the ZBASIC compiler will evaluate an expression like the one above. Since I indicated in the header that arithmetic expressions are to be optimized to integer, the first thing that happens is that the PEEK WORD is added to the PEEK in *16 bit integer*. That means that the maximum value that can possibly be returned is 65535. Hence, files in excess of 64K are not handled accurately.

However, as indicated on p.38 in the MATH section of the ZBasic manual, one way to force the compiler to use floating point math is to include a decimal point.

I knew that.

No typo in Greg's letter, then, he simply knew what he was doing. The FILE_LEN bug is not terribly serious (unless you want to use AWP2TXT on really BIG files), but it will

undoubtedly show up the day you really NEED perfection.

I wish that was all, but I made one more flub. In the OPEN_FILE long function, I had the following line:

```
BUFFER% = &8900 - (FNUM * &400)
:REM get 1K I/O buffer for ProDOS
```

This is the buffer address for the 64K version of ZBasic (i.e. &8900). Since everthing else in the program was designed for the 128K version, it would be best to change this value to &AC00. You know, this may explain some of the problems I've been having with memory management... Thanks again, Greg.

Mr. ZBasic (Greg) also wondered what I was getting at when I suggested using SET_MARK to enable you to BLOAD any arbitrary chunk of a file. His BLOAD function does allow you to grab as much of a file as you want, but only as long as you want to start with the 0th byte (i.e. from the beginning). SET_MARK allows you to move up the "starting" byte to any point within the file.

Even though Apple Computers, Inc. is enriched by Greg's presence, it is obvious that we fans of ZBasic lost a powerful and knowledgable supporter. In spite of his new responsibilities, Greg has continued to offer occassional insights and tips. Okay, everybody mail a couple dollars to Apple Computer, Inc., Mail Stop 27-S, Cupertino, CA...

Last month we printed SUPER.INPUT, an INPUT routine with several powerful editing capabilities and features. At that time I pointed out that, with a very few changes, it could be turned into a full screen editor. Well, "...a very few changes" is a pretty subjective phrase, but this month I'd like to offer up a very simple full screen editor based on SUPER.INPUT. Again, I have endeavored to keep it quite generic in flavor so that it is of use on virtually any computer. I have used variables for hardware specific memory locations and noted their purpose, hence changes should be fairly easy.

The most significant new concept in SCREEN.INPUT is the use of an element in the INDEX$ array to handle each screen line. You could potentially use a standard string array, but ZBasic's INDEX$ array has some features that make it particular useful for this type of application.

First and most important, the INDEX$ array can be selectively cleared; that is, you can zero it out in one quick statement without touching any of your other data. If you need to reuse the editing routines several times within a program, you can do so quickly and easily.

Additionally, using INDEX$ leaves room for future features. Since the INDEX$ array has super quick element insertions and deletions, it is a simple matter to add "delete line" and "insert line" commands.

The only other new concept is the notion of "line records". For a variety of reasons, it is useful to have an array whose elements hold information about the state of the matching screen line. I call this array LINEREC%(x). Its elements contain the length of each screen line and whether or not that line terminates in a carriage return. A line with no carriage return has a line record equal to the length of the line only, a line terminating with a carriage return has a line record equal to the length of the line plus 128.

Therefore, a line like...

This is line X.<CR>

...has a line record equal to 143 (a length of 15 plus 128 for the CR). Since the longest of lines is never more than 80 characters, this is a safe way to compress our data a tad.

We have also arranged our code so that each new line is assumed to have a carriage return at the end. If the line wraps to a new line, the carriage return flag is removed from the old line. In this way we create a mechanism where a carriage return marks the end of a paragraph, which is what I was really after, anyway.

SCREEN.INPUT keeps all of the old features from SUPER.INPUT, but has three new features over its single line cousin. Number one is the use of up and down arrow keys. These keys have been defined as variables according to their ASCII values as mapped onto the Apple II. Again, redefining them for your computer should be fairly painless.

The second spiffy feature is automatic wordwrap, which I'll explain when we get there.

The final new feature is the ability to work within any given window or rectangle on the text screen. If you want SCREEN.INPUT to work its magic within a nice little rectangle you've plopped on the screen (like say, the COMMENT field of a database), just pass SCREEN.INPUT the horizontal and vertical coordinates of the upper left and lower right corners. It will reset the text window and clear out the new rectangle.

There are a few things SCREEN.INPUT does NOT do. It does NOT have commands for the automatic insertion or deletion of entire lines, although this is quite easy to add (c.f. p.239 of the ZBasic manual).

It also does not pull down text when you do a carriage return in the middle of text on a screen line. Instead, it just moves the cursor down a line and flush against the left margin. Again, this feature is easy to add (I've done it myself in ***ProTools***' Mr.Ed), but I leave it as the proverbial "excercise for the reader".

Let's dig in...

```
REM ------------------------------
REM Screen Input
REM
REM by Ross W. Lambert
REM Editor, Znews
REM Copyright (C) 1989
REM ------------------------------
:
:
:
REM ------------------------------------------------------
REM              DIM arrays and strings
REM ------------------------------------------------------
:
:
CLEAR 1920 : REM reserve enough room in INDEX$ array for a screen full
DIM LINEREC% (24),C$,1 K$,BS$,SP$,CR$,RIGHTARROW$,UPARROW$,DOWNARROW$
DIM 1 ESC$,INV$,NORM$,DELETE$,CHAR$,160 TL$,TR$,NEW$
DIM 40 ZTITLE$,90 ZHORZLIN$
:
:
REM ------------------------------------------------------
REM           Define variables and constants
REM ------------------------------------------------------
:
:
BLINKRATE = 90 : REM rate of cursor flash (lower means faster)
L = 0 : REM input line number (for INDEX$ array 0 - 23)
TN = 0 : REM total number of lines
WPOS = 1 : REM cursor position in input line
CURSORFLAG = 1 :REM displaying cursor or character? (1 = cursor / 0 = character)
INSERTFLAG = 0 :REM insert mode flag
CURSOR$ = CHR$(127):REM default cursor is checkerboard
BS$ = CHR$(8):REM backspace key
```

```
SP$ = CHR$(32) : REM blank space
CR$ = CHR$(13):REM carriage return
RIGHTARROW$ = CHR$(21): REM right arrow key
UPARROW$ = CHR$(11)
DOWNARROW$ = CHR$(10)
ESC$ = CHR$(27): REM escape key
ENDFLAG = 1 : REM are we entering text at end or in middle?
DELETE$ = CHR$(127): REM delete key
CRFLAG = 0 : REM are we pushing along a carriage return?
:
REM The memory locations listed define the text window for the Apple II.
REM MS-DOS can use DEF PAGE statement instead (c.f. later in the function)
:
WINDOWLEFT = 32
WINDOWRIGHT = 33
WINDOWTOP = 34
WINDOWBOT = 35
:
REM ------------------------------------------------
REM              Define Functions
REM ------------------------------------------------
:
:
:
LONG FN INCLEN(LN) :REM increment line record quickly
  LINEREC%(LN) = LINEREC%(LN) + 1
END FN
:
LONG FN LINEREC(LN)    :REM update a line record
  LONG IF LINEREC%(LN) => 128          :REM add 128 if CR on line
    LINEREC%(LN) = LEN(INDEX$(LN)) + 128
  XELSE  :REM else make it length of line only
    LINEREC%(LN) = LEN(INDEX$(LN))
  END IF
END FN
:
:
LONG FN XLINE(LN)    :REM mask off CR flag to get true length
  LONG IF LINEREC%(LN) => 128
    XL = LINEREC%(LN) XOR 128
  XELSE
    XL = LINEREC%(LN)
  END IF
END FN = XL
:
LONG FN UPDATE_ALL      :REM recompute all line records
  FOR X = 0 TO TN
    FN LINEREC(X)
  NEXT
END FN
:
LONG FN BEEP       : REM I got sick of typing PRINT CHR$(7);
  PRINT CHR$(7);
END FN
```

The code in the functions above could potentially be put into subroutines. However, if you add features to this editor I expect that you'll end up putting them back into function form. The reason for this is that it some operations require fiddling with the line records attached to many different lines. It would be a pain in the 'ol algorithm to have to adjust the current line variable (L) all the time before a GOSUB. If left as long functions, you can leave the current line count the same and just pass some other variable with the number of the line you want to adjust.

Note that the funcion UPDATE_ALL is never called. I included it to help you if you ever expand this editor. Imagine that you've reformatted a large section of text due to a block delete (a reasonably tough function to add - be forewarned). In such a case many (if not all) the line records would need to be updated. UPDATE_ALL does the job quickly and easily.

With the exception of the code that sets up and clears the current text window, this next section should look somewhat similar to last month's SUPER.INPUT. MS-DOSsers don't forget to leave out the Apple II POKEs. Y'all get to use the handy DEF PAGE command.

```
:
REM -----------------------------------
REM Screen Input
REM -----------------------------------
:
LONG FN SCREEN_INPUT (XTOP,YTOP,XBOT,YBOT)
  CLEAR INDEX$
  LINELENGTH = XBOT - XTOP
  YBOT = YBOT - 1 : REM can't write on bottom of window
  :
  REM MS-DOS folks substitute DEF PAGE XTOP,YTOP TO XBOT,YBOT
  :
  POKE WINDOWLEFT,XTOP
  POKE WINDOWRIGHT,XBOT-XTOP+1:REM Most computers just POKE WINDOWRIGHT,XBOT
                               REM The Apple II is odd in this respect
  POKE WINDOWTOP,YTOP
  POKE WINDOWBOT,YBOT
  :
  :
  LOCATE 0,YTOP
  CLS PAGE
  L = 0  : REM init current line
  TN = 0 : REM init total line count
  :
  REM GetKey updates total line count, displays cursor, and scans keyoard
  :
  "GetKey"
  :
  LOCATE WPOS-1,YTOP + L     : REM position cursor
  :
  IF L > TN THEN TN = L
  :
  LONG IF L = TN AND WPOS > FN XLINE(L) : REM at end of text?
    ENDFLAG = 1
  XELSE
    ENDFLAG = 0
  END IF
  :
```

```
   LONG IF ENDFLAG = 0 AND WPOS <= FN XLINE(L) : REM cursor past end of text on a line?
     CHAR$ = MID$ (INDEX$(L),WPOS,1)  : REM figure character under cursor
   XELSE
     CHAR$ = SP$
   END IF
   :
   :
   "keyscan"
   :
   DO
     C$ = INKEY$
     IF CURSORFLAG THEN PRINT CURSOR$;BS$; ELSE PRINT CHAR$;BS$;
     LONG IF LEN(C$) = 0  :REM no key pressed?
       BLINK = BLINK + 1  :REM inc flash counter
       LONG IF BLINK = BLINKRATE:REM time for a change?
         BLINK = 0 :REM reset counter
         IF CURSORFLAG THEN CURSORFLAG = 0 ELSE CURSORFLAG = 1 : REM change what's
showing
       END IF
     END IF
   UNTIL LEN(C$) :REM loop until a key is pressed
   :
   PRINT CHAR$;BS$; :REM restore character under cursor
   :
   REM ----------------------------------------
   REM       Add text to current line
   REM ----------------------------------------
   :
   : REM branch away if other than normal text
   :
   IF C$ < SP$ OR C$ = CHR$(127) THEN "Special_Chars"
   :
   REM if starting new line and at end of all text, mark record with CR
   :
   IF WPOS = 1 AND ENDFLAG = 1 AND LINEREC%(L) < 128 THEN LINEREC%(L) = LINEREC%(L) +
128
   :
   LONG IF ENDFLAG = 1    : REM at end of text?
     "TackOn"
     INDEX$(L) = INDEX$(L) + C$     : REM just tack keystroke on end
     FN INCLEN (L)
     WPOS = WPOS + 1
     PRINT C$;
     IF FN XLINE(L) => LINELENGTH THEN GOSUB "Wordwrap"
     GOTO "GetKey"
     :
   XELSE  : REM mid-text
     LONG IF INSERTFLAG = 0 : REM overwrite mode
       LONG IF WPOS > FN XLINE (L) : REM past end of text on line?
         GOTO "TackOn"
       XELSE :REM if inside text on a line
         IF WPOS > 1 THEN TL$ = LEFT$(INDEX$(L),WPOS-1) ELSE TL$ = "": REM get text
left of cursor
         TR$ = RIGHT$(INDEX$(L),FN XLINE(L) - WPOS) :REM chop ahead of cursor
```

```
      INDEX$(L) = ""
      INDEX$(L) = TL$ + C$ + TR$:REM overwrite char under cursor
      PRINT C$;
      WPOS = WPOS + 1
      GOTO "GetKey"
    END IF
    :
  XELSE :REM insertflag = 1
    IF WPOS > 1 THEN TL$ = LEFT$(INDEX$(L),WPOS - 1) ELSE TL$ = ""
    TR$ = RIGHT$(INDEX$(L),FN XLINE(L) - (WPOS-1))
    WPOS = WPOS + 1
    TR$ = C$ + TR$
    INDEX$(L) = TL$ + TR$
    FN LINEREC(L)
    :
    FN LINEREC(L)  :REM recount chars in case of change
    LONG IF FN XLINE(L) > LINELENGTH : REM too many characters?
     LONG IF RIGHT$(INDEX$(L),1) = SP$ : REM is last char a space?
      INDEX$(L) = LEFT$(INDEX$(L),FN XLINE(L)-1) : REM chop space if so
     END IF
    XELSE : REM room for word as is
      PRINT TR$;
    END IF
    GOTO "GetKey"
  END IF
END IF
:
:
REM ----------------------------
"Special_Chars"
REM ----------------------------
:
IF C$ = ESC$ THEN "Exit" :REM  ESCape
:
LONG IF C$ = DOWNARROW$ :REM  Down arrow
  IF YTOP + L + 1 => YBOT THEN FN BEEP:GOTO "GetKey"
  :
  L = L + 1
  IF L > TN THEN LINEREC%(L) = LINEREC%(L) + 128
  IF WPOS-1 > FN XLINE(L) THEN GOSUB "PadSP"
  GOTO "GetKey"
END IF
:
:
LONG IF C$ = UPARROW$
  IF YTOP + L - 1 < YTOP THEN FN BEEP:GOTO "GetKey"
  L = L - 1
  IF WPOS-1 > FN XLINE(L) THEN GOSUB "PadSP"
  GOTO "GetKey"
END IF
:
:
LONG IF C$ = RIGHTARROW$ :REM Right arrow pressed
  LONG IF WPOS < LINELENGTH:REM at end of line?
    WPOS = WPOS + 1 :REM move right one character
```

```
    IF WPOS-1 > FN XLINE(L) THEN GOSUB "PadSP"
    GOTO "GetKey"
    :
  XELSE  :REM wrap to next line
    REM if cannot wrap, beep and leave
    IF YTOP + L + 1 => YBOT THEN FN BEEP:GOTO "GetKey"
    :
    WPOS = 1
    L = L + 1
    GOTO "GetKey"
  END IF
END IF
:
:
LONG IF C$ = BS$    :REM back space key
  LONG IF WPOS = 1 :REM first char?
    LONG IF L > 0  :REM first line?
      L = L - 1
      WPOS = LINELENGTH + 1
      GOTO "GetKey"
      :
     XELSE
       FN BEEP:GOTO "GetKey"
     END IF
     :
   XELSE  :REM wpos > 1
     WPOS = WPOS - 1:REM just move left one character
     GOTO "GetKey"
   END IF
 END IF
 :
 LONG IF C$ = DELETE$ :REM delete
   :
   LONG IF WPOS = 1 :REM at beginning of line ?
     FN BEEP : GOTO "GetKey"
   XELSE : REM mid-line
     :
     WPOS = WPOS - 1
     TL$ = LEFT$(INDEX$(L),WPOS-1) :REM chop line left of cursor
     LONG IF WPOS-1 < FN XLINE(L) : REM within text?
       TR$ = RIGHT$(INDEX$(L),FN XLINE(L) - WPOS)
       INDEX$(L) = TL$ + TR$
     XELSE
       TR$ = ""
     END IF
     :
     FN LINEREC(L)
     LOCATE 0,YTOP + L : PRINT INDEX$(L);: CLS LINE
     GOTO "GetKey"
   END IF
 END IF
 :
 LONG IF C$ = CR$        :REM carriage return
   REM too many lines already?
   IF YTOP + L + 1 => YBOT THEN FN BEEP:GOTO "GetKey"
```

```
    L = L + 1
    WPOS = 1
    GOTO "GetKey"
  END IF
  :
  LONG IF C$ = CHR$(2) :REM go to beginning of current line
    WPOS = 1
  END IF
  :
  LONG IF C$ = CHR$(14)  :REM go to end of current line
    WPOS = FN XLINE (L) + 1
  END IF
  :
  LONG IF C$ = CHR$(25)  :REM delete to end of line
    CLS LINE
    INDEX$(L) = LEFT$(INDEX$(L),WPOS - 1)
    FN LINEREC(L)      :REM kill text to end and update
  END IF
  :
  LONG IF C$ = CHR$(5)  :REM exchange cursors
    LONG IF CURSOR$ = CHR$(127)
      CURSOR$ = CHR$(95)
      INSERTFLAG = 1
    XELSE
      CURSOR$ = CHR$(127)
      INSERTFLAG = 0
    END IF
  END IF
  :
  GOTO "GetKey"
  :
  :
```

You'll notice the subroutine "PadSP" below is called whenever the arrow keys cause the cursor to wander onto a line past any text. This code pads the current line with spaces out to the cursor position. This is necessary given the method I used to add characters to the current line - they're just glued on to the end. If the user were to cursor to the middle of a blank line and then type an X, the X would really end up as the first character on that line if "PadSP" was not allowed to work. Fortunately for me, ZBasic appends the spaces quickly enough that this approach works.

```
REM ------------------------------
REM          Subroutines
REM ------------------------------
:
:
"PadSP" :REM fill out line to cursor with spaces
FOR X = FN XLINE(L) TO WPOS - 1
  INDEX$(L) = INDEX$(L) + SP$
  FN INCLEN(L)
NEXT
RETURN
:
```

The wordwrapping routine deserves a little attention since it is new.

First, the routine checks to see if there is another line available in the window. If not, no wrap is possible so we beep and skip town.

Next, the routine examines the last character typed. If it is a space, no word wrapping is necessary. The only adjustment needed is to kill the carriage return flag on the line we just finished. We then RETURN.

If the last character at the end of the current line is NOT a space, the "Scan4space" loop scans backwards until we do find a space. A space, by the way, is the only character we accept as a word break. If you want to create a super sophisticated editor, you could amend this to include hyphens, embedded "break-it-here" markers, etc.

If "Scan4space" cannot turn up a space on the line, it does a pretty brutal thing; it jumps to the "WordTooLong" subroutine. "WordTooLong" clears your entire line and makes you type it over. It adds insult to injury with a beep. I would have it display an error message except that I have no way to know in advance where your program would want to display the message. Since this function is designed to work inside of a larger body of code, you'll need to handle this error in a manner consistent with your program. I guess that's a nice way of saying you're on your own.

If our little word wrapping routine makes it past "Scan4space", it does the obvious: it rips out the last word from the line we just finished and adds it to the line below. If the line below already exists, the word to be wrapped overwrites the beginning of the next line. A more sophisticated routine would not overwrite if the program was in insert mode. Yes, that's another excercise for the reader... But hey, who would've figured that the source code to a flexible full window editor would ever fit into the pages of a small monthly newsletter, anyway?

Anyone acquainted with the power of ZBasic, that's who.

```
REM ------------------------------------------------------------
"Wordwrap"   :REM pull down word to next line if too long
REM ------------------------------------------------------------
:
IF YTOP + L + 1 => YBOT THEN FN BEEP:RETURN  :REM if no more line, beep
:
LONG IF C$ = SP$ : REM last char a space?
  WPOS = 1 : L = L + 1 : REM move to next line
  IF LINEREC%(L-1) > 128 THEN LINEREC%(L-1) = LINEREC%(L-1) - 128
  RETURN
END IF
:
"Scan4space"
X = 0
DO
  X = X + 1
  TL$ = RIGHT$(INDEX$(L),X) :REM scan backwards for a space
  LINEREC%(L) = LINEREC%(L) - 1
  TR$ = LEFT$(TL$,1)
  IF FN XLINE(L) = 1 THEN GOSUB "WordTooLong":RETURN
UNTIL TR$ = SP$ AND FN XLINE(L) < LINELENGTH  :REM & make sure line will fit
```

```
  :
  INDEX$(L) = LEFT$(INDEX$(L), FN XLINE(L) )  :REM subtract word from line
  :
  TL$ = RIGHT$(TL$,LEN(TL$) - 1)    :REM chop leading space
  LOCATE FN XLINE(L),YTOP + L:CLS LINE :REM clear to eol
  L = L + 1  :REM now fix next line
  :
  LINEREC%(L-1) = LINEREC%(L-1) - 128
  :
  INDEX$(L) = RIGHT$(INDEX$(L),FN XLINE(L) - LEN(TL$)) : REM make room
  INDEX$(L) = TL$ + INDEX$(L)
  FN LINEREC(L)
  WPOS = LEN (TL$) + 1
  LOCATE 0,YTOP + L : PRINT INDEX$(L);
  :
  RETURN
  :
  "WordTooLong"
  INDEX$(L) = ""
  FN LINEREC%(L)
  LOCATE 0,YTOP + L
  CLS LINE
  WPOS = 1
  FN BEEP
  RETURN
  :
  "Exit"
END FN
:
REM demo
:
MODE 2
PRINT CHR$(15); : REM makes text window white for demo purposes
FN SCREEN_INPUT (5,5,40,10)
PRINT CHR$(14); : REM back to normal
MODE 2
:
FOR X = 0 TO TN
  PRINT LINEREC%(X)"  "INDEX$(X)  : REM reveals what you typed for fun
NEXT
PRINT:PRINT"WPOS:"WPOS;" TN: "TN;" L:"L
END
```

Automated Appending Action for All

By Mohawk Man

Jay Jennings (aka Mohawk Man) is our MS-DOS specialist. A former radio disk jockey, Jay has written radio station management software, adventure game shells, and a host of other goodies. Like me, he lives in the wilds of Alaska. Unlike me, though, he has a deviant desire to relocate to the parking lots and 7-11's of Southern California.

One of the things that computers are supposed to do is to make our life easier. They're supposed to take the drudgery out of otherwise dreary tasks. Unfortunately, they sometimes introduce new dreary tasks while taking care of the old ones.

One of the nicest things about ZBasic is being able to use long functions. One of the worst things about ZBasic is that those nifty functions have to go at the beginning of a file. I got very tired of using RENUM to open up enough room to APPEND in a new function, so I sat about to remedy that. In the process I discovered a bug in the way files are handled in the Apple version of ZBasic. We'll get to that in a moment.

What does FILEMAKR do? It takes any number of TXT files (like long functions) and creates one large file for you. You can then take that file and add your code onto the end of it. No more APPENDing!

First, a warning about the program. It's very unfriendly. But because of that, it's also short and simple. You can jazz it up later if you want. (Don't hit me for throwing in a bit of advertising, but a totally hip version of this program, complete with mouse support and more, will be in the next version of ***ProTools***.) The program as it stands works very well for appending text files together, though.

Editor: Jay is working feverishly to finish the MS-DOS version of ***ProTools****. I expect them to be finished shortly after you get this newsletter. Right, Jay?*

The DIM statement in line 50 will only allow you to choose 10 files. You can change that to whatever number suits you. The rest of the program is basically self explanatory.

THE APPLE BUG

I haven't figured out all the variables in this bug, but in a nutshell, you need to make sure that the first file you open in a program is designated file number one. If you open a file as number two, then open a second file as number one, your first file (2) will take some of it's characteristics from the second (1) file. Confused? So was I...for hours.

Here's how I found it: I opened a file for OUTPUT (OPEN "O", 2, FILE1$) as file number two. Then I opened a file for INPUT (OPEN "I", 1, FILE2$) as file number one. I started reading from file 1 and writing to file 2. Then I got a strange error..."End of file error in 2."

Wait a minute! How do you get that error in an OUTPUT file?

It turns out that ZBasic thought that the file designated as number two (the output file) was as long as the file designated as one (the input file, opened second). When that many bytes had been written to the output file, an error resulted.

How to get around this? Simple. Just open file 1 first, file 2 second, etc. If possible, you could just open all your files at the beginning of your program and leave them open until you're done with them.

```
:
REM /* FILEMAKR.ZBS */
REM /* Another Mohawk Man Creation */
REM /* Copyright 1989 - PunkWare */
```

```
:
DIM FILE$(10)
:
MODE 2 : CLS
:
PRINT "Enter the file names to append.
Hit RETURN alone when done."
DO
   TOTFILES = TOTFILES + 1
   INPUT "File to append:
";FILE$(TOTFILES)
   IF FILE$(TOTFILES) = "" THEN TOTFILES
= TOTFILES - 1 : QUIT = 1
UNTIL QUIT
:
PRINT : PRINT
PRINT "Enter the file to write to. Hit
RETURN alone to quit."
INPUT "Output File: ";OUTPUT$
IF OUTPUT$ = "" THEN END
:
OPEN "O", 1, OUTPUT$
:
PRINT : PRINT : PRINT
FOR FILENUM = 1 TO TOTFILES
   PRINT "Processing file:
";FILE$(FILENUM)
  GOSUB "COPY"
NEXT
:
CLOSE : END
:
"COPY"
ON ERROR GOSUB 65535
OPEN "I", 2, FILE$(FILENUM)
IF ERROR GOSUB "ERROR"
DO
  LINE INPUT  2, A$
  PRINT  1, A$
UNTIL ERROR <> 0
:
ERROR = 0
PRINT  1, ":"
CLOSE  2
RETURN
:
"ERROR"
PRINT : PRINT "Error: ";ERRMSG$(ERROR)
INPUT "<C>ontinue or <S>top? ";B$
B$ = UCASE$(B$)
IF B$ = "C" THEN ERROR = 0 : RETURN
STOP
```

Inversing an IBM

By Jay Jennings

Using the IBM in text mode is great. All those nice colors are available to spice up your screen. For those of you who have never programmed any other computers, that's not a normal occurrence. Most systems have one color for the text and a second for the background. Period. Coming from the Apple II world, I was amazed at the things you can put on the text screen.

Then I took a look at the ZBasic commands for using color on the text screens and I was just a tad confused. If BASICA can use normal numbers for screen colors, why can't we? Why should we have to mess with individual bits in a byte?

It was that thought, and the need to inverse a block of text, that led me to create two long functions for ZBasic. The first, FINDCOLOR, will return the foreground and background colors for a specified screen location. The second, SETCOLOR, works like the normal COLOR statement. With it you can use the normal color numbers (1-7) to specify the screen colors. Using the two functions together, you can inverse a block of text.

All you have to do to use FINDCOLOR is to pass it the X and Y coordinates of the area of the screen to check. It will return the value of the foreground in ZFGC and the value of the background in ZBGC. You can then swap those values and use them as the parameters for SETCOLOR to create an inverse color.

These two functions aren't earth shattering, but they do save time. And they're much easier to use than the chart in the back of the ZBasic reference manual.

```
 REM /* FINDCOLR.FN */
 REM /* Another Mohawk Man Creation */
 REM /* Copyright 1989 - PunkWare */
 :
 REM /* Finds the screen attributes at
the specified screen coordinates. */
 :
 LONG FN FINDCOLOR(ZX,ZY)
   ZA = SCREEN(ZY,ZX,1)
   ZFGC = ZA MOD 16
   ZBGC = ((ZA - ZFGC) / 16) MOD 128
 END FN
 :
 REM /* This allows you to use normal
color numbers (0-7) in text mode */
 :
 LONG FN SETCOLOR(ZFGC,ZBGC)
   COLOR ,ZFGC + (ZBGC << 4)
 END FN
 :
 REM /* Sample code showing calling
syntax */
 :
 MODE 2 : CLS
 FN SETCOLOR(3,1) : CLS
 PRINT "This is light blue text on a dark
blue background."
 FN FINDCOLOR(10,10) : OLDFGC = ZFGC :
OLDBGC = ZBGC
 FN SETCOLOR(ZBGC,ZFGC) : REM /* reverse
the colors */
 PRINT "And now we've inverted the colors
so it's dark blue on light blue."
 FN SETCOLOR(OLDFGC,OLDBGC) : REM /* use
the original colors again */
 PRINT "Finally, we're back to normal."
 :
```

Ask Mike Rochip

Dear Mike,

Being new to computing, and trying to learn to program on my own, without benefit of formal training, is about as hard as Chinese arithmetic. I chose ZBasic in an attempt to avoid the complicated process of compiling. The package performs beautifully, and all would be well, except; with my lack of background trying to make some of the conversions from other BASICs to ZBasic is next to impossible. In particular, I'm having trouble saving sprite figures to disk, and then recalling them to the screen. I've been trying to accomplish this with the ZBasic package for about a year with no success.

It would be of great help to us (beginners - I've been programming for about about 2 years) if you could publish some code in the newsletter on the handling of partial screen files.

Any help that you could give me along these lines would be greatly appreciated.

Good luck in your new venture, I'll be looking forward to the newsletter and the quarterly disks.

Thank you,

Edgel Hall
Seaford, Delaware

Edgel -

I sincerely wish I'd started *Znews* about a year ago. I think a monthly would've saved all of us some frustration.

Nevertheless, I'm not certain that I can help you because you left some important pieces of information out of your letter. Everyone take notes, now...

First, always include the make and model of the computer on which you are using ZBasic.

Second, always include the version of ZBasic you are using.

Third, always include the version of the operating system you are using (e.g. MS-DOS 3.0, or ProDOS v 1.6, etc.). There are programmer's bugs, ZBasic bugs (of which

there are very few), and operating system bugs (of which there are usually a veritable plethora). If that were not bad enough, there are even hardware bugs on some machines (maybe most). It's a wonder anything works, I sometimes mutter to meself.

Fourth, always include the names of the other inhabitants of your computer's chips. On the IBM and compatibles this would include TSRs (Terminate and Stay Resident programs), auto-exec batch files used, and control system files (Ugh, I'm IBM illiterate so I hope I got those terms correct - RWL.). On an Apple IIGS it would include desk accessories and INIT files. On a Z-80 it would include nothing I know about... which brings up another subpoint. Is there any Z-80 & CP/M gonzo hacker sorta person out there looking for a technical editorship? The pay is rotten, the work is very part time (maybe ten hours a month or so) but pretty interesting, and the prestige is (you fill in the blank). Access to a modem is required, BTW.

In this case, Edgel, you should have also told me the name of the BASIC you were trying to translate and the machine it was running on.

Nevertheless, I'll try to shed some light on the situation.

The only time I've used sprites is when I did a little work on the Commodore 64 and C-128. As implemented on those computers, a sprite is a graphics object controlled and displayed by a special dedicated video chip. Commodore BASIC might include commands to control sprites (I don't remember for sure), but ZBasic does not. In fact, to my knowledge ZBasic does not run on any machines that support sprites. Thus, on the one hand, my answer to your question is that it cannot be done.

My other hand, however, instructs me to tell you that there are many ways around this buggaboo. If you're working with an MS-DOS machine, you have ZBasic's powerful GET and PUT commands working in your favor.

They can save and restore any arbitrary section of a graphics screen. If you draw an object, you can capture it and redraw it anywhere. And since GET and PUT save the data into an array, you can also write that data out to disk.

If you're working with the Apple II, GET and PUT are not supported. I suspect this was primarily due to the cramped confines of memory on this machine. Saving graphics takes a LOT of memory (just ask a IIGS or Macintosh programmer). But the Apple has other alternatives. Zedcor included a DRAW/XDRAW function which displays "vector graphics", otherwise known as shape tables. A shape table is a collection of the "instructions" necessary to recreate and draw a shape. The table of numbers in the table stand for instructions like, "Move one unit over and plot a point, then move up and plot another point", etc. The advantage to shape tables is that they can be scaled and rotated easily. The disadvantage is that they are S-L-O-W.

An alternative is to use "bitmapped" shapes which are shoveled right into screen memory (this is the way GET and PUT work). I hate to sound like a billboard, but ***ProTools*** for the Apple II has a pair of assembly language functions that will do the trick quite quickly. The new functions (we call them SAVESCREEN and RESTORESCREEN) are basically the ZBasic GET and PUT commands implemented for the II.

A cheaper but slower alternative for the II is to use the ZBasic POINT command. Using point and a FOR-NEXT loop, you can inquire about the status of any range of points on the graphics screen. You can save this data in an array and POKE it back into screen memory to make moves or changes.

I know this information is pretty general, but I'll wait until I get some more specific information from you before I attempt to provide some sample code.

-MIKE ROCHIP

This is as good a place as any to highlight a couple of slight changes in policy for Ariel Publishing. Up to this point, all source code printed within the pages of our publications (annoying alliteration, ain't it?) have been declared public domain.

For our own protection, we are now copyrighting everything. This does NOT mean that you may not freely use our code in commercial products. We just ask that you fill out a license agreement before you do. There is no charge for this, and there are no grievous requirements or hidden pitfalls. We just need to know who is doing what with our stuff. This goes for the routines and functions in ***ProTools***, too.

The only stipulation we stick you with is a requirement to make some sort of acknowledgment in your documentation or title screen. Something like, "The AppleWorks file reading routines courtesy of Znews" would be quite sufficient.

The other slight change in our policies is that we will not distribute the quarterly disk to folks who do not subscribe to the newsletter. I cannot imagine why anyone would want one without the other, anyway. The quarterly disk is a service for newsletter subscribers. We distribute them at our cost, for the most part. Compare our disk prices to those of other publications and I think you'll see what I mean.

Znews

Copyright (C) 1989 by Ross W. Lambert
and Ariel Publishing, Inc.
All Rights Reserved

Subscription prices in US dollars effective April 15, 1989:
1 yr..$35, 2 yrs..$65 CAN & MEX add $9, other non-USA add $15

Back issues are available at $3.00 each.

WARRANTY AND LIMITATION OF LIABILITY

I warrant that the information in Znews is correct and somewhat useful to somebody somewhere. Any subscriber may ask for a full refund of their last subscription payment at any time. MY LIABILITY FOR ERRORS AND OMISSIONS IS LIMITED TO THIS PUBLICATION'S PURCHASE PRICE. In no case shall I or my contributors be liable for any incidental or consequential damages, nor for ANY damages in excess of the fees paid by a subscriber.

Please direct all correspondence to:

Ariel Publishing, Inc.
P.O. Box 266
Unalakaleet, Alaska 99684 USA

Znews is a product of the United States of America.
ZBasic is a registered trademark of Zedcor, Inc.

BULK RATE
U.S. POSTAGE
PAID
SEATTLE, WASH.
PERMIT NO. 74